A2: Classes and Facades

Re-submit Assignment

Due Sep 8 by 5pm **Points** 100 **Submitting** a file upload

Assignment Overview

The main goal of the assignment is to wrap a fairly clunky C-style library for pdf creation into a nicer-to-use class. This is an example of the facade design pattern discussed in class.

You will also make a small class to generate spirals and then tie together your facade class and the spiral class in an application to turn text into spiral art.

The application will use a Makefile to build so you will practice this style of build management.

Getting Started

Download the Haru free PDF library from

http://libharu.org (http://libharu.org/)

I followed a link to GitHub from there and then did a

$ git clone **https://github.com/libharu/libharu.git** (https://github.com/libharu/libharu.git)

to clone it in my file system on the lab2 machines. One other machine types you can either do that, or go to the GitHub site and clone it from there.

As can often be expected from casual open-source projects, you will find that documentation is spotty, inconsistent, or outdated. Pay attention to where you are looking for information and interpret material flexibly. For example, there is also an older SourceForge page for this project with some out of date information.

Move into the top-level libharu folder. On the command line, type

$ cmake .
$ make

I skipped a 'make install' as we do not have admin privileges. This should build a .a static library in the src directory. Confirm it is there. We can leave the include files in the include folder and the .a static library in src/ and direct our Makefile to look for them there.

The Project

Start with the pdfExample.cpp file linked in Canvas under the course website. This is really a stripped-down version of one of the demo files in the haru library. Get this to compile. You will want to use -I (capital i) to show where the include files are and -L to show where the haru library is and -l (lowercase l) to give the name of the library. This page

http://libharu.sourceforge.net/compile_your_program.html (http://libharu.sourceforge.net/compile_your_program.html)

has some suggestions on compiling, except that we are not using the PNG library (and the haru library name is libhpdfs.a with an 's').

If you run the program, it generates a pdf with text in a circle. You can view this pdf from the command line with "okular filename.pdf".

For this assignment, you will need to make the text go in a spiral, instead. All of your code needed for the requirements below should be in a folder called "A2" for assignment 2. You will eventually zip up this folder to submit it.

Requirements

Put a comment block on top of each code file with your name and CS3505 and the assignment.

A C++ Haru Facade Class

Make a C++ facade class called HaruPDF that holds the data necessary for Haru to work and that provides class methods to access enough haru functionality to produce your spiral page. Your facade class should provide a simple interface to the haru library. Your class will mostly be a reorganization of the pdfExample.cpp haru library calls.

The HaruPDF class should not have any spiral generating code it in - the haru class should be general purpose enough for setting up a document, placing text on a page, and saving the pdf that it can support the pdf needs of the pdfExample code. It should not be monolithic - the methods of the class should support the pdfExample with a few steps, not one call to HaruPDF::makeSpiralPDFDocumentFrom("hello").

The code for the wrapper class must be saved in HaruPDF.h and HaruPDF.cpp. The .h file should only provide the interface and all implementation should be done in HaruPDF.cpp. The grade for this portion will be primarily on your architecture of the class and basic functionality. Is it a class someone else could use easily?

The Haru library uses a weird error mechanism and many functions return a STATUS flag. Let's ignore this stuff until we get a chance to discuss C++ error mechanisms. The example code I provide doesn't use any of those return values.

A Spiral Class

The example code has some rough computations for placing text in a circle. The key elements are an x,y position and two angles, one the rotation about the center and the other a 90 degree turned angle that aims the text characters.

Make a Spiral class that:

1. Has a single constructor that takes in a
    1. center of spiral coordinate: double centerX, double centerY
    2. a starting radius double. The spiral should grow larger from this start size in a clockwise direction and start at "12 o'clock" on the page. The code may internally choose the growth rate and the rate at which the spiral turns. You must enforce a positive starting radius and may enforce a minimum radius to preserve the appearance of the text - clamp the start values to these minimums and continue.
    3. a starting angle double, given in degrees. The spiral should start at the angle clockwise from 12 o'clock on the page.
2. An overloaded operator++ which internally advances the spiral to its next position. Operator++ is curious in that it handles two versions, ++obj and obj++. Research how to do this and implement both. The next position should be suitable for the next character of the text to be placed. This next position should be pleasing: text should not overlap or be too spread out and the lines should have some spacing. This pleasant spacing should at least work for spirals that fit on one 8.5 x 11in page.
3. Getters for the x,y text position, spiral angle, and text angle called getTextX, getTextY, getSpiralAngle, and getTextAngle, all of which should return a double. Angles should be reported as degrees. The TextX and TextY are the position on the page of the current angle point, the SpiralAngle is the total clockwise rotation from 12 o'clock for the current point, and the TextAngle is the angle a letter should be printed at for the current spiral point. The TextAngle is just a 90 degree rotation of SpiralAngle. You will probably not use the SpiralAngle getter in your test code.
4. The spiral class should use C++ math library pi consts, not your own.
5. An overloaded ostream << function which reports in a nicely formatted way the state (such as angle and radius) of a Spiral object. Rely on users of this function to provide any desired newlines after outputting the spiral object.

The Spiral class and the << function should be in a file Spiral.cpp and Spiral.h and have implementation code in the .cpp file. The Spiral class should not have any haru functionality, although it should share the haru sense of coordinate systems and scale (in other words, you do not need to figure out the units of both and make them consistent).

A Test Program

Write a test program spiralPDF.cpp and executable spiralPDF, borrowing from the structure of the example file provided, which makes a nice spiral of text. The text should come from a command line argument to the program, so that a command like

$ ./spiralPDF "This is my sample text"

would make a spiral out of the "This is my sample text" phrase (without the quotes). The text should be pleasingly placed along the spiral. The text can be clipped by the page boundaries. The program should make a pdf file called spiralPDF.pdf. You will have to view the pdf with another program (double-clicking the file from the file browser opens a viewer on the lab2 machines) or there are command-line viewers such as okular.

The test program should report an error if no sample text is provided as a command line argument and then quit.

A Makefile

Add a Makefile for the project. It should know how to make spiralPDF and any .o files from your code as well as a 'clean' target which removes all .o files and the executable. A 'test' target should build as necessary and run ./spiralPDF with an example text string input. You should set a variable LIBHARU in the Makefile to give the position of the Haru library top-level (the folder that has src and include inside it) and use that variable when setting library and include paths in the Makefile. We will be overriding that variable setting to compile your program. You should not have any absolute paths, such as /home/frank/myCode in the Makefile.

The code should compile and run on the lab2 eng.utah.edu machines. g++ should be configured with options -Wall and -std=c++11. Any warnings from compilation should be fixed in the code before submitting.

Generate a PDF

Execute your code with a demonstration phrase and keep the generated pdf in the A2 folder.

Submission

Zip your A2 folder and submit through Canvas. Do not include libharu in your submission.

Peer Review

Make the code professional looking. I plan on having small peer reviews of the code and I will discuss that aspect later. The reviews are assigned at the due the Monday after review assignment.

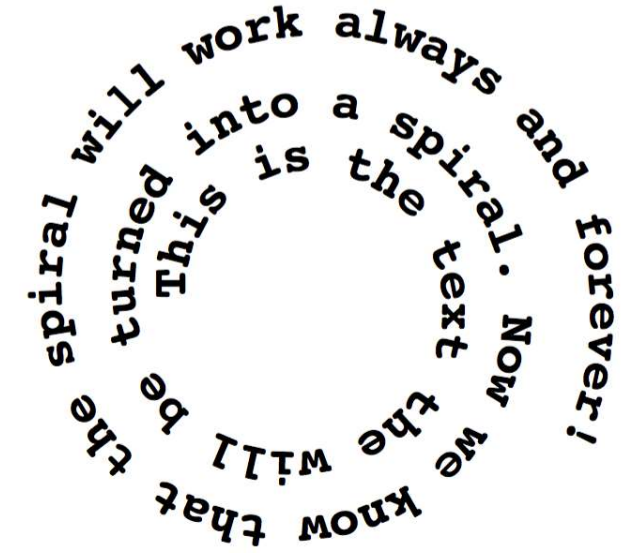
Suggestions

First, if you manage to download libharu and compile it and use the provided code to compile an executable, proceed by making incremental changes to the project so that you always have a working example.

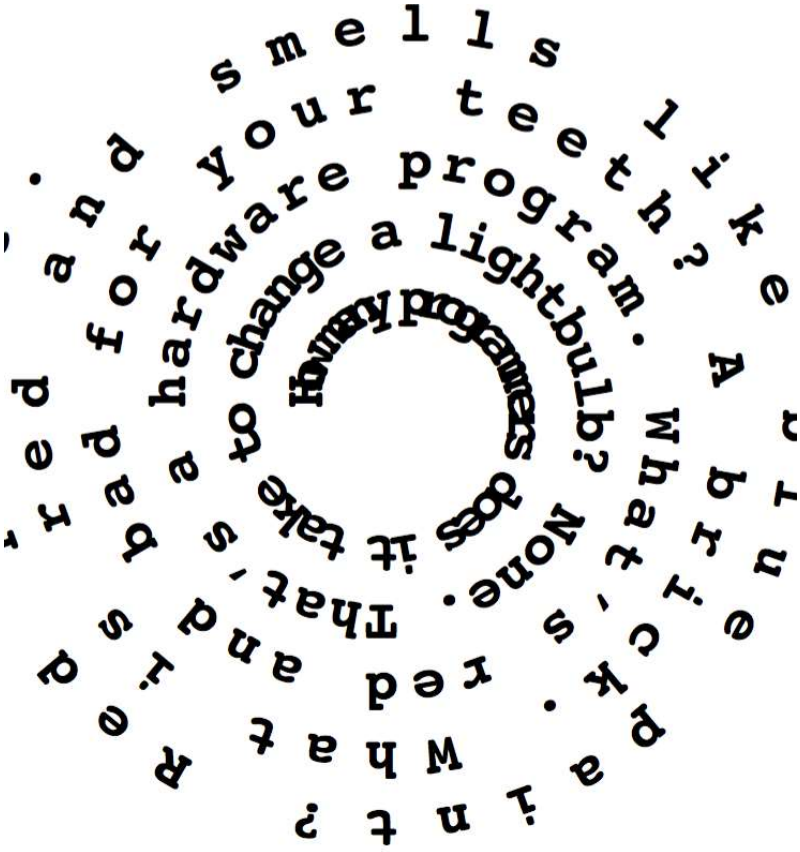
Second, do not worry too much about what libharu is doing. Much of your code will be class methods that call the libharu functions, but you really do not need to worry about the details - just so long as it works! You do need to think about the interface you are providing to libharu. Does it capture an elegant or minimal way of controlling the behavior of libharu? And, of course, remember that you are only making a class that does enough to place text for the spiral - not the whole haru library.

Finally, do not get hung up on the spiral code. You can make a spiral by making the radius bigger as you go around in a circle. Many of you will make logarithmic spirals, some of you may try an Archimedean spiral. The difficulty is that a constant change in angle results in close-spaced letters in the center of the spiral and far-spaced further out. You need to work to make the letters roughly evenly spaced, at least for a spiral that fits on a page. The further out you go, the smaller the angle between letters needs to be.

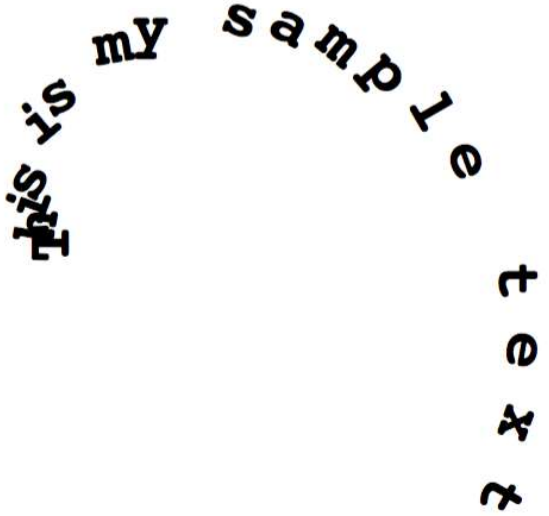
Here are some examples. This first one looks nice. It would get full credit for the spiral and is better than needed.



Here is one with some problems. The inner part is obviously overlapping and the outer part has too much space between characters. To be honest, if this person has just expanded the spiral faster and started at a bigger inner radius, those edge cases wouldn't have come into play and just the pretty good middle part would be visible.



This last one has a lot of problems - I think it starts at a zero radius. The real issue here is that the test string is too short to really exercise the spiral and that is a different problem from some bad math. Make good test cases!


this is my sample text